Architectures and Mechanisms to Maintain efficiently Consistency in Collaborative Virtual Environments

Cédric Fleury, Thierry Duval, Valérie Gouranton, Bruno Arnaldi

Architectures and Mechanisms to Maintain efficiently Consistency in Collaborative Virtual Environments

Cédric Fleury*
INSA de Rennes
IRISA, UMR CNRS 6074
Université Européenne de Bretagne, France

Thierry Duval†
Université de Rennes I
IRISA, UMR CNRS 6074
Université Européenne de Bretagne, France

Valérie Gouranton‡
INSA de Rennes
IRISA, UMR CNRS 6074
Université Européenne de Bretagne, France

Bruno Arnaldi§
INSA de Rennes
IRISA, UMR CNRS 6074
Université Européenne de Bretagne, France

ABSTRACT

Collaborative virtual environments (CVE) enable users to collaborate and interact with each other by sharing a common virtual environment. Consistency of the virtual environment is very important to provide an efficient collaboration between users. However, users sharing a CVE may be scattered over different physical locations, so CVE systems have to guarantee the consistency of the virtual environment despite network issues such as low bandwidth or network latency. Absolute consistency is nearly impossible to achieve because it would prejudice the responsiveness of the system during users' interactions. So, CVE systems have to deal with a trade-off between consistency and responsiveness of the system. This paper presents a detailed survey of architectures and mechanisms used to improve the consistency of a shared virtual environment. Architectures of CVE systems are studied according to their impact on the consistency. Contrary to previous state of the art reports which classify CVE systems according to the network connections or to the data distributions, we choose to examine these two properties separately. This classification enables us to deal with the increasing number of hybrid architectures which mix different architectural choices to meet their requirements. Consistency maintenance mechanisms are also examined. First, a time synchronization has to be achieved in order to enable users to have the same state of the virtual environment at the same time. Second, the virtual environment can be seen as a database shared by several users, so CVE systems have to manage users' concurrent access to the objects of the virtual environment. Finally, we discuss the need to enable CVE systems to adapt themselves to performance constraints induced by the use of mainstream network such as DSL Internet access or by security requirements of industrial users.

Index Terms: H.5.3 [Information Interfaces and Presentation (e.g. HCI)]: Group and Organization Interfaces—Computer-supported cooperative work; C.2.4 [Computer-Communication Networks]: Distributed Systems—Distributed applications; I.3.7 [Computer Graphics]: 3-Dimensional Graphics and Realism—Virtual reality

1 INTRODUCTION

One of the main goals of collaborative virtual environments is to enable users to work together in a natural way and to provide them a truly interactive experience. Generally, each user uses her own computer in order to have individual interaction capabilities or to meet the others if they are not located at the same geographical place. So, this collaborative work has to be achieved over a local area network (LAN) or a wide area network (WAN). For example, in the case of the French ANR project CollaViz[1], remote experts have to analyze together scientific data using an Internet connection through a secured proxy. Such network connections have a strong impact on the consistency of the shared virtual environment because they delay the transmission of the modifications of the virtual environment. Moreover, while some users of a CVE can interact through immersive devices, powerful computers, and high-bandwidth network connections, some other users can share the same CVE trough simple workstations and low-bandwidth network connections. Even if some users have a slow network connection or a not powerful computer, they must have the same state of the virtual environment than the other users. Ensuring the consistency of the CVE is the best way to enable users to have an efficient collaboration, because it can avoid conflicts between several users' actions, or misunderstandings when users perform collaborative tasks.

To define the consistency of a CVE, Delaney et al. [5] explain that a CVE has be considered as a distributed database with users modifying it in real-time. The consistency maintenance is to ensure that this database is the same for all users, i.e. users observe or interact with the same data. The consistency of a CVE can be characterized by the following three criteria:

- **synchronization:** (1) time synchronization: an event (modification of the state of a shared virtual environment) should happen simultaneously on all the simulation nodes (users' computers). (2) spatial synchronization: the CVE objects should be located as precisely as possible for all users.
- **causality:** events order has to be the same for all users.
- **concurrency:** conflicts can occur when users change the same parameter of a virtual object at the same time. These conflicts have to be managed to avoid that users make their own modifications and have inconsistent states of the CVE.

Consistency is directly linked to system responsiveness. Responsiveness can be defined as the time needed by the system to respond to users' actions. Responsiveness during interactions can be quantified by the system latency, i.e. the time between a user's action and the response of the system. This latency is often due to the transmission time over the network and to the processing delays of the events. Improving the consistency of a CVE can increase latency in interactions and vice versa. So, CVE systems must reach a compromise between consistency and system responsiveness.

Latency is especially a problem for applications that require lots of interactions. Delaney et al. [5] present several values of latency found in the literature. It seems that no consensus has been found about these values. However, the maximum latency values fluctuate between 40 and 300 ms to maintain real-time interaction, and a maximum latency of 100 ms seems sufficient for most of the applications. These values depend also on the jitter (the variance of the latency) of the system. Delaney et al. [5] explain that jitter has a more significant impact on users' performance than latency. It would be better to have a quite high and constant latency (i.e. with a low jitter), rather than a lower latency with a higher jitter.

*e-mail: cedric.fleury@irisa.fr
†e-mail: thierry.duval@irisa.fr
‡e-mail: valerie.gouranton@irisa.fr
§e-mail: bruno.arnaldi@irisa.fr

[1] www.collaviz.org

In this paper, Section 2 describes the different architectures of CVE systems and their impact on consistency and system responsiveness. Then section 3 examines the consistency maintenance mechanisms used in CVE systems. Finally, section 4 concludes and discusses what can be improved to maintain the consistency in CVE systems that use mainstream networks and need secured network connections.

2 System Architecture

Consistency and performance of a CVE are highly correlated with its system architecture. For example, some architectures maintain a strong consistency of the CVE but introduce latency in interactions. In contrast, other architectures accept a few inconsistencies between each user's state of the virtual environment, but enable users to have a better responsiveness during interactions. Previous state of the art reports classify CVE systems according only to how the simulation nodes (users' computers) are connected together [6, 10] or only how data are distributed on these nodes [13, 7]. Although these two characteristics are often interrelated, we chose to use these two criteria separately for our classification: the kind of network architecture (part 2.1) and the kind of data distribution (part 2.2). This classification choice is motivated by the fact that more and more hybrid solutions mix different architectural choices to meet their requirements. As different communication protocols can be used to communicate information during a simulation, we choose this third criterion to complete our classification (part 2.3).

2.1 Network Architecture

A CVE system is usually made up of several interconnected nodes that can be geographically scattered. Each node can communicate with the others through three main data transmission methods:

- *unicast:* transmission only from one node to another,
- *broadcast:* transmission from one node to all the others,
- *mutlicast:* transmission from one node to a subset of other nodes.

Delaney et al. [6] distinguish two basic network organizations used for CVE systems: the peer-to-peer architecture and the client/server architecture. They also introduce hybrid architectures that combine these two solutions.

2.1.1 Peer-to-peer architecture

The peer-to-peer architecture enables fast communications between pairs of users, because events are transmitted directly from one simulation node to another one's (see Figure 1). So, it enables a few users to have strong synchronization, and consequently close interactions. However, when the number of users increases, the number of messages transmitted on the network becomes considerable, especially when an *unicast* transmission is used (if the simulation contains N members, a node has to send N-1 messages to transmit one event). Consequently it is difficult to contact all the users at the same time to transmit modifications of the virtual environment. So, time synchronization and global consistency of the CVE can be difficult to ensure. This kind of architecture appeared with the first CVE systems (Reality Build for Two [2] which connects only two computers, MR Toolkit [19]) and is used in military applications (SIMNET [4], NPSNET [14]).

2.1.2 Client/server architecture

This kind of architecture is based on a central server. All the users connect themselves to this server (see Figure 2). The central server manages the communication between different users and store data. This kind of architecture enables the server to contact all the users at the same time. So time synchronization and consistency of the CVE are easier to maintain than in the peer-to-peer architecture.

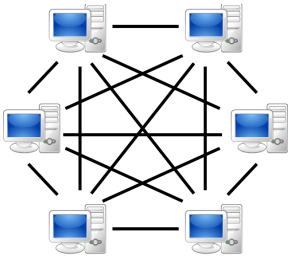

Figure 1: peer-to-peer architecture

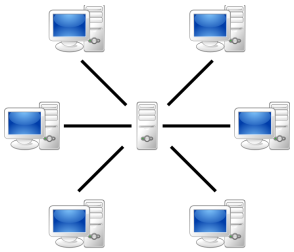

Figure 2: client/server architecture

However, when two users want to interact together, all the communications have to pass through the server, which increases interaction latency. Moreover, when the number of users increases, a bottleneck can occur on the server due to too many communication requests. So all the communications can be slowed down. For example, a client-server architecture is used in: RING [9], BrickNet [20], and ShareX3D [11].

2.1.3 Hybrid Architecture

To overcome the previous limitations, some systems use an hybrid network architecture that uses both peer-to-peer connections and one or several servers. For example, it is possible to speed up communications between users using peer-to-peer connections or to maintain a better consistency with a server.

In SPLINE [21], several servers share up-to-date information (messages, events, etc.) with peer-to-peer connections between these servers (see Figure 3). At the beginning of the session, a session manager connects users to one of these servers. Then users only communicate with their assigned server. On the one hand, this solution avoids the bottleneck on the server when the number of users increases. On the other hand, it enables to easily connect users with slower connections or secured connections. Indeed, each server can perform additional processing such as compression or communication with a specific protocol. However, using too many servers can increase the latency of the system and the load of the servers.

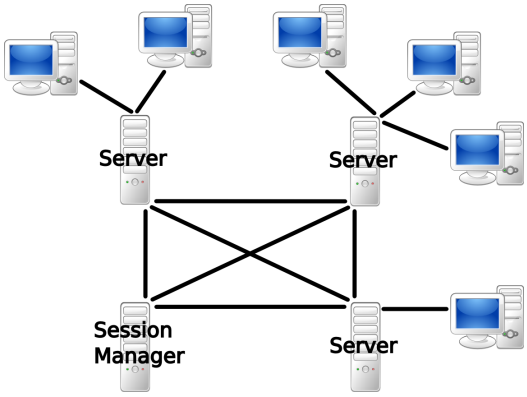

Figure 3: several servers used peer-to-peer connections

Anthes et al. [1] suggest another hybrid architecture to facilitate collaboration between nearby users (according to their location in the virtual environment) by reducing the latency between them (see Figure 4). Users are connected to the CVE through a server. When users are close in the virtual environment, temporary peer-to-peer connections are established between these users to increase the consistency of the virtual environment between them.

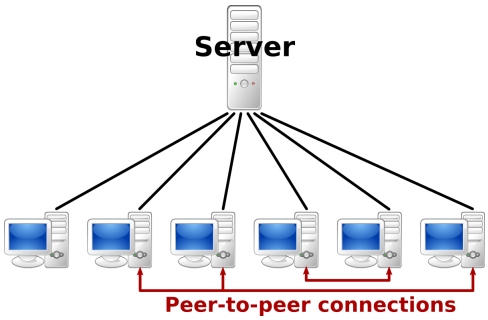

Figure 4: temporary peer-to-peer connections between close users

The OpenMASK platform [15] architecture is also an hybrid one. Indeed, it uses peer-to-peer connections to send updates and events between users, but it uses also a server to manage the identification of virtual objects and to add dynamically users during a simulation.

2.2 Data Distribution

As stated by Macedonia et al. [13], choosing the location of the virtual environment data (i.e. geometric data, textures, etc.) is a critical decision when designing a CVE system. We must determine which computers store this data, but also which computers execute the processing related to each virtual object. We distinguish three data distribution modes: the shared centralized world, the homogeneously replicated world and the partially replicated world (distributed world).

2.2.1 Shared centralized world

Some systems such as Vistel [23] use one database shared by all users. Data in relationship with the virtual environment are stored on a central server. Similarly, behaviors of the CVE objects are executed on this server (see Figure 5(a)). Users simply connect themselves to the server for participating to the simulation (this approach requires a client/server network architecture). When a user wants to modify an object state, she sends a request to the server. The server processes the modification request, then transmits the up-to-date state to all users, including the one who asked for this modification (see Figure 5(b)).

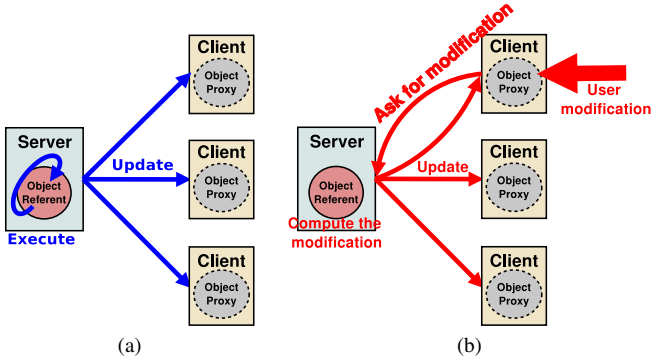

Figure 5: (a) executions of object behaviors and (b) object modifications in shared centralized world

This method ensures consistency between all users and avoids data replication. However, this solution has two main drawbacks:

- Interaction latency can increase when transmission delays occur between the clients and the server. Indeed, each modification request has to pass through the server before returning to the user (see Figure 5(b)). This lack of responsiveness during interactions can be annoying for users.
- If the number of users is high, a bottleneck can appear on the server because it has to send updates to all users at the same time (especially with *unicast* connections).

2.2.2 Homogeneous replicated world

This kind of data distribution is used in many CVE systems (SIMNET [4], MR Toolkit [19]). The state of all the simulation nodes (users' computers) is initialized with the same database that contains all the information about the virtual environment (terrain, geometric models, textures, object behaviors, etc.). Data can already be present on the node at the user connection (such as in most of the video games). Otherwise, data has to be replicated from a server, or from the other nodes already present in the simulation. During the simulation, the database evolves independently on each node and all object behaviors are executed locally (see Figure 6(a)). Additionally, a synchronization mechanism can be used to control executions of object behaviors on each node. To maintain consistency, only changes of object states and some special events (collision between two objects, etc.) are transmitted on the network in order to enable all users to update their database (see Figure 6(b)).

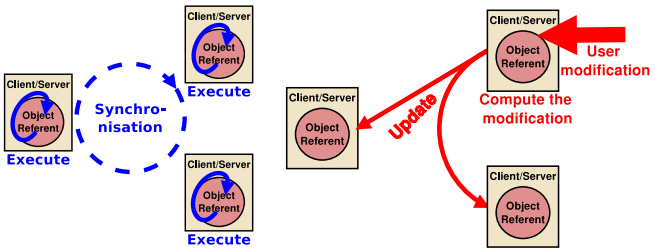

Figure 6: (a) executions of object behaviors and (b) object modifications in a homogeneous replicated world

This data distribution has two main advantages:

- The number and the size of messages transmitted on the network are really small because only update messages are sent.
- The latency is very low when a user interacts. In fact, modifications of the virtual objects are performed locally before being sent to the other users using update messages.

However, data replication has also some drawbacks:

- Data replication can introduce inconsistencies between each user's virtual environment because of delays or losses during the transmission of update messages.
- We need additional mechanisms to manage the concurrent access on each node. Indeed, a user can perform a local modification of an object, but modification conflicts can only be checked when the modification is transmitted to the other users.
- If the amount of data is large, the database on each node will be large. So this solution is not really appropriate for very large data sets such as scientific data or complex CAD models.
- This approach is not really flexible, especially if users want to add new objects that the initial database does not know.

2.2.3 Partially replicated world (or distributed world)

Some CVE systems choose hybrid solutions between totally centralized and totally replicated data distributions in order to avoid the drawbacks of these two solutions. These hybrid solutions distribute the data and their treatments between the simulation nodes. It reduces the necessary resources and eases

the consistency maintenance. In the literature, these hybrid solutions are called partially replicated world or distributed world. We present some hybrid solutions that seem particularly interesting.

To avoid the duplication of all the data in all nodes, data can be distributed on the network between these nodes. So the database can be seen as a shared and distributed memory. In DIVE [8], when a new user connects herself to the collaborative session, she replicates only the necessary objects on her node instead of downloading the whole data of the virtual environment (see Figure 7). However, if this user needs other objects during the simulation, she has to re-download them dynamically.

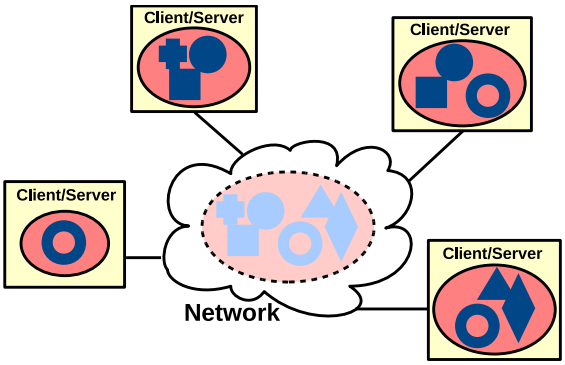

Figure 7: Data partially replicated on each simulation node

DIVE uses peer-to-peer connections to manage communications between users (transmissions of messages or object data when it is necessary). Moreover, DIVE uses a server to backup the data present on all the nodes. It enables to save the state of the virtual environment in case of user disconnections or to restart the simulation where it had been left during the previous session. This method enables to have a high number of users and a very large amount of data without necessarily duplicate this data on each node. However, dynamic transmission of data and consistency maintenance induce a high cost of communications between users. When the state of an object is changed, update messages must be sent to all the users even if they do not own this object. Indeed, a user cannot know who owns the modified object.

The main difficulty of this partially replicated world is to efficiently replicate the missing data without disturbing the users' interactions. According to Lee et al. [12], two solutions can be used:

- The prioritized transfer: this technique consists in selecting first the objects that are in the user field of view, and transfering these objects using level of detail (LODs) or multiresolution techniques. This solution maximizes graphical fidelity of the virtual environment as well as interactive performances by mediating graphical details and transmission overhead of the objects.
- Caching and prefetching the data: this technique consits in caching and prefetching the data that users will probably need. It enables to make it immediately available when users ask for it. However, this solution must predict efficiently which objects will interest first the users in order to define a loading priority. Generally, the distance between users and objects is used to determine this priority, assuming that users will be interested first by the closest objects. Other solutions suggest to add the direction of displacement of users or to use the object type to determine the scope of interest of users. However, when the number of objects becomes large, this prediction is more difficult to achieve.

To reduce the cost of communications for the updates, BrickNet [20] uses a server to keep information about the shared objects: access rights on these objects, which users own each object, etc. The virtual environment of each user can be different because she can add shared, but also private objects. Object behaviors are executed on each node. BrickNet uses a client/server architecture in which the server manages the communications between users. For example, if a user wants to share one of her private objects, she has to notify the server that the object becomes now shared. Then the server keeps a list of users who ask for this object. When a user wants to modify an object, she must first ask the server for the modification rights on this object (only one user can modify an object at the same time). When her request is granted, this object can be modified locally. Then the new state of the object is transmitted to the users who have this object using a list of object owners on the server. In this approach, the server enables to ease the consistency maintenance and to manage the concurrent access.

In OpenMask [15], data of the virtual environment are replicated on each node. However, each object behavior is executed only on one node. To achieve this, OpenMask [7] uses a referent/proxies paradigm. The referent is assigned to a particular node and defines the object behavior. On the other nodes, proxies are created to represent the object. A proxy has the same interface as the remote referent (same inputs, same outputs, etc.). However, it computes no processing locally and it is forced to "follow" the referent behavior (see Figure 8(a)). The OpenMask kernel [15] maintains the consistency between the referent and its proxies. When a user manipulates an object with the referent on her node, first the object is modified locally, second an update is sent to all the other users (see Figure 8(b)). However, if the referent of the manipulated object is not on her node, the modification of the object is computed first on the remote node which owns the referent. Then this node transmits updates to all the users, including the user who has asked for the modification (see Figure 8(c)). In this second scenario, transmission delays on the network can introduce latency in interactions. However, this solution enables to combine the processing power of all the simulation nodes. It enables also to ensure a better consistency of the virtual environment and to manage implicitly the concurrent access to the objects (in a first step, only the referent can be modified).

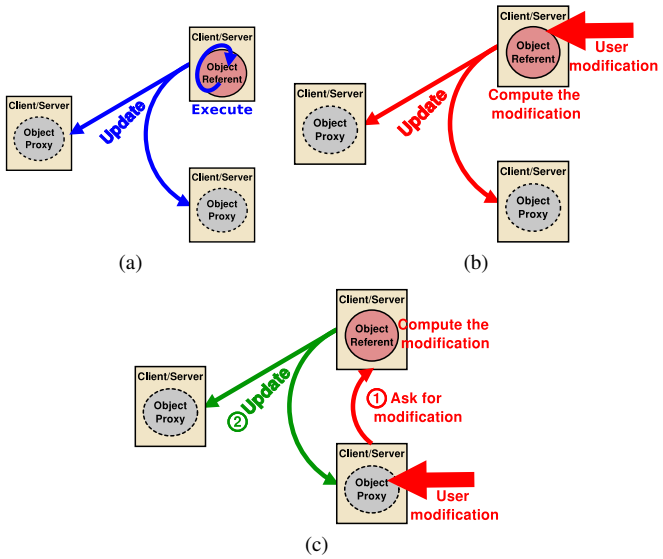

Figure 8: (a) executions of object behaviors, (b) object modifications when user has the referent, and (c) object modifications when user has a proxy in OpenMask

2.3 Communication protocols

Several protocols can be used to communicate the information necessary for consistency maintenance between the simulation nodes. A protocol describes how the nodes will communicate on the network. Generally, a network connection can be decomposed in three layers: the network layer, the transport layer, and the application layer. Usually, protocols for CVE systems are defined either in the application layer or in the transport layer. The most commonly used network layer protocol is IP (Internet Protocol).

2.3.1 Classical Protocols

TCP (Transport Control Protocol) and UDP (User Datagram Protocol) are the most commonly used transport layer protocols. TCP ensures reliable transmissions between two nodes using a system of acknowledgment and retransmission. TCP can achieve only *unicast* transmissions. On the contrary, UDP sends data in a non-connected mode: there is no way to verify if packets are correctly received and in which order. However, it can be useful to quickly send data to several users (using *broadcast* or *multicast* transmissions). Indeed, a new packet can be sent without having to wait that all the previous packets have been received by all the concerned users. To solve the reliability problems of UDP, SPLINE [21] and SIMNET [4] send update messages with the whole state of the object (not only the modifications). If some messages are lost, an up-to-date object state can be restored when a new message arrives. So there is no need of retransmission.

According to Delaney et al. [6], the choice of the transport layer protocol depends of the kind of interactions in the CVE. For punctual interactions which involve a transmission of a small amount of information or an irregular transmission, it is better to choose UDP. On the contrary, for persistent interactions which require a transmission of a large amount of information or a regular transmission, the TCP protocol is most commonly used.

2.3.2 *Multicast* oriented Protocols

The first works on CVE have chosen to use transmissions based on *unicast* or *broadcast*. For example, MR Toolkit [19] uses *unicast* transmissions between nodes widely distributed on the Internet. However, these solutions are not efficient for a very large number of users. Moreover, as in SIMNET [4], *broadcast* transmissions may be problematic in the case of lots of network connected nodes because it will submerge the network with unwanted data. For all these reasons, researchers have proposed several solutions using *multicast* protocols.

To achieve *multicast* transmissions, a solution is to use the *multicast* IP network layer, which enables to send messages simultaneously to many recipients (on a non-reliable way). However, according to Delaney et al. [6], this solution has several drawbacks:

- it can be difficult to implement efficiently on a point-to-point medium,
- many Internet routers are not able to support *multicast*,
- the number of user groups can be limited because of various issues such as addressing, congestion control, or administration.

To overcome these problems, network overlays have been developed to provide IP *multicast* capability over networks that do not offer *multicast* capability at the network layer. DIVE [8] and NPSNET [14] use the "MBone" (*multicast* backbone) which creates a virtual network providing *multicast* with encapsulations of *multicast* packets in normal IP packets.

2.3.3 Virtual Reality dedicated Protocols

Some other application layer protocols use the transport and network layer to optimize the transmissions of application-specific data. El Zammar [7] lists some application layer protocols dedicated to virtual reality applications:

Real-Time Protocol (RTP) provides network transport functionalities for applications transmitting real-time data such as streaming video, audio, or simulation data, both in *unicast* or in *multicast*. Mauve et al. [17] propose a new protocol RTP\I which adapts RTP for interactive applications. RTP\I uses four types of packets to manage event communications, objects states transmissions, state changes, state queries.

Virtual Reality Transfer Protocol (VRTP) is designed to be a support for the VRML (Virtual Reality Modeling Language) in the same way that HTTP is the HTML support. In other words, VRTP can be seen as an extension of the HTTP protocol to meet the requirements of CVE, because HTTP is not sufficient to manage 3D interactive objects. The VRTP approach proposes that a node can take the role of a client, a server, or a peer if needed. This node can be seen as a client examining databases of other users, as a server responding to requests from other users, and as a peer participating in a simulation with groups of users who speak through *multicast* channels.

Distributed Worlds Transfer Protocol (DWTP [3]) is a specific application protocol for sharing virtual environments over the Internet. It is based on standard protocols such as TCP/IP and UDP/IP. It enables to transport several types of data: events, messages, files and data streams. Events are used to ensure the consistency of the virtual environment between the users. The messages are predefined events such as join or leave the virtual environment. A file can be a 3D scene description, an avatar or an object geometry. It requires a reliable transport. Data streaming transmits a continuous data stream such as audio or video. This data type does not require a reliable transmission. The DWTP concept is based on daemons and participants. The role of the daemon is to provide services to the participants:

- the reliability daemon: detects transmission failures (for unreliable protocols like UDP).
- the retransmission daemon: transfers the lost data by using *unicast* connections.
- the world daemon: transmits the virtual environment content to the new participants.
- the *unicast* daemon: extends the architecture for participants who do not have a *multicast* channel.

Distributed Interactive Simulation (DIS) comes from the SIMNET military simulation platform [4]. The DIS standard (IEEE 1278) defines specific messages called PDU (Protocol Data Units) which communicate specific information between the simulation nodes. A few PDU are dedicated to users' interactions with the virtual environment and the others are used to transmit various data and actions on the virtual environment such as the apparitions of objects, the use of weapons, explosions, etc. However, DIS is very specific to the military simulation context because the message format is fixed in the protocol for specific military simulation objects. In order to extend this protocol for non-military applications, a new protocol version called High Level Architecture (HLA) has been proposed.

Interactive Sharing Transfer Protocol (ISTP [22]) has been developed in the context of SPLINE [21]. However, ISTP can be used with other CVE systems. Each ISTP process acts as a client and a server at once. To communicate information about virtual objects, ISTP is based on sub-protocols:

- the connection protocol: used to establish TCP connections between two ISTP processes.
- the object state transmission protocol: used to transmit the state of the virtual object using *unicast* or *multicast* UDP connections.
- the data streaming protocol: used for the communication data streams such as audio (based on RTP).

2.3.4 Industrial Environment Specific Protocols

For some industrial uses, classical or VR dedicated protocols are not well adapted to modern security constraints. For example, ShareX3D [11] has to deal with firewalls that support only the HTTP protocol. So ShareX3D proposes to use the "long polling" technique for server-to-client and client-to-server-to-client communications (ShareX3D uses a client/server architecture). This technique is used to overcome the fact that HTTP does not allow requests from the server to the client. The "long polling" can be divided in four steps:

- The client sends a connection request to the server.
- As long as the server has no event to send to the client, the connection is kept open.
- Once the server has an event to send to the client, it transmits it with the open connection. Then the connection is automatically closed.
- The client immediately re-sends a new connection request to the server and so on.

Time and processing power necessary to establish a new connection after sending each message can introduce latency in users' interactions, when events need to be sent with an high frequency rates.

3 Consistency Maintenance Mechanisms

In addition to the system architecture and the communication protocol, some mechanisms can be used to improve the consistency of the virtual environment. First, we present time synchronization mechanisms which enable each user to process modifications of the CVE at the same time. Second, we examine the different solutions used to manage the concurrent access to virtual objects.

3.1 Synchronization

Time is a fundamental element of a CVE. However, the notion of time can differ from one application to another. Delaney et al. [5] distinguish two different notions of time in the CVE:

- **Absolute time:** the time of a CVE can be based on a clock periodically synchronized between each node based on the coordinated universal time (UTC).
- **Logical or causal time:** the time of a CVE can be based on a logical clock which is not precisely synchronized between each node. This time can be seen as an ordered event sequence. When no new event occurs, it stays the same.

The relationship between time and consistency is very important. In a perfectly consistent CVE, all users have the same global state at the same absolute time. However, this perfect case can never append because of network latency. Delaney et al. [5] list different solutions to improve consistency over time according to the interaction responsiveness required.

3.1.1 Lockstep Synchronization

The lockstep synchronization used in RING [9] or OpenMask [15] is the easiest way to ensure consistency of a CVE. It consists in stopping users until all of them have processed the current simulation step. So each user does not increment her logical clock until all the other users have acknowledged that they are ready for the next simulation step. This solution imposes that events are processed in the correct order and avoids roll-backs. However, it guarantees consistency but not real-time consistency. If there are delays or losses during transmissions, the waiting time of users increases and the system responsiveness is reduced. Moreover, the simulation step are not necessarily constant and so the system jitter can be substantial.

3.1.2 Imposed Global Consistency

This technique consists in delaying the processing of both local and remote events with a fixed time in all nodes. This time is usually defined according to the maximum values of the network latency. This delay enables to increase the probability that the remote events are received before processing all the events of a simulation step. This method enables to have a strong global consistency between all the users with an absolute clock. However, this solution introduces interaction latency which value varies according to the network characteristics. However, this latency is constant during all the simulation.

3.1.3 Delayed Global Consistency

Contrary to the imposed global consistency, the goal of this technique is to maintain an asynchronous consistency. Users perceive the same state of the virtual environment, but at different time. Each event is marked with a "timestamp" (using a logical clock). The logical clock is maintained on each node. So the state of a CVE can be rebuilt locally with the good order of events. However, this delayed consistency can disturb the collaboration tasks (users may not have the same virtual environment at the same time).

3.1.4 Time Warp Synchronization

"Time Warp" synchronization is an optimist technique, which consists in processing each event as soon as it arrives. The events are also marked with a "timestamp". When an event is received with a older "timestamp" than the event which has just been executed, the mechanism must cancel the processing of all events which have a most recent "timestamp" (roll-back). Then it run again all these events to catch up with the current time. Moreover, it must also send messages to cancel incorrect messages sent during the wrong execution (roll-back propagation).

This synchronization method enables users to have low latency interactions. However, it can only be used when roll-backs happen rarely, because their are extremely annoying for users. Some systems propose to quickly display several key-frames during the re-execution of events to facilitate the users' understanding. Finally, this method requires to store the received events to re-execute them in case of roll-back.

3.1.5 Predictive Time Management

This method proposes to predict events and to send them on the network before they occur. This concept was first proposed by Roberts et al. [18] in the PARADE system to manage the consistency when users collaborate through a network with inherent latency. Obviously, this mechanism can not be applied to all the objects of the virtual environment because some objects are not predictable. Particularly, user's actions are difficult to predict. For example, PARADE uses this method for the collision detection.

Events are predicted locally, marked with a "timestamp" and sent to other nodes, where these events will be executed at the appropriate time (defined by the "timestamp"). This predictive manage-

ment is interesting only if the time between the sending of the predicted event and its processing is higher than the network latency. Otherwise the message will arrive too late to be processed. If the prediction is false, the system needs to make roll-backs to resolve mistakes. To minimize roll-backs, PARADE proposes to send predicted events just-in-time by using estimations of network delays. To achieve this, PARADE determines the network delays by studying the RTT (Round-Trip Time) of the network using specific messages.

3.1.6 Server Synchronization

In client/server architectures, the server can be used to manage an event synchronization using a logical clock. In ShareX3D [11], the server maintains a "state number" for each object of the CVE. When the server receives a modification for an object, it increases the "state number" of this object. Users maintain also a "state number" for each object corresponding to the last update messages received for this object. When a user asks the server for new modifications about an object (see "long polling" in 2.3.4), she sends the local "state number" for this object. If the user's "state number" is the same than the server one, the user is up-to-date and her request stays in standy. However, if the user's "state number" is older than the server one, the server sends to her an update message for the object and the new "state number" value.

This solution provides a simple way to ensure that users have the most recent state of an object. However, it does not ensure that all events will be received and processed by users. This is not an issue in ShareX3D because the server sends the whole state of objects, so users can restore teh state of an object with only one update message.

3.2 Concurrency control

The centralized data distribution ensures an implicit control of concurrent access to the CVE objects, because this data can be changed only on the server. It is the same for systems which use a referent/proxies paradigm as OpenMask [15], because only the referent of an object can be modified by users. However, when data is distributed on each node (homogeneous or partially replicated worlds), the users can access and modify objects locally before these changes are transmitted to other users. So it is necessary to manage explicitly the users' concurrent access to these objects to avoid inconsistencies in the virtual environment. When an object can be modified by only one user at the same time (non-collaborative interactions), Lee et al. [12] distinguish three kinds of mechanisms to manage the concurrent access:

- **pessimistic mode**, as in BrickNet [20]: This mode ensures that only one user can modify an object at the same time with a lock system. When a user wants to manipulate an object, she asks to become its owner. An object have only one owner. So if the object have already an owner, the user have to wait until this owner releases the ownership of this object. Only the current owner of an object can modified its parameters. In this way, no concurrent access to an object can occur. However, when the network latency or the number of users are high, the necessary time to acquire an object ownership can be long and therefore introduces latency in interactions. BrickNet [20] uses the server to save which user is the owner of each object. However, this mode is difficult to set up in the case of a peer-to-peer architecture. Indeed, when a user requests the ownership of an object, she must ask all the other users if they own this object.
- **optimistic mode**: This mode enables users to modify objects without checking the potential concurrent access on these objects. So users can have low latency interactions with these objects. However, when a conflict occurs, it is necessary to make a correction. It can be complex and requires also that users perform again their actions.
- **prediction based mode**, as in PARADE [18] or ATLAS [12]: This mode consists in predicting for each object which users may manipulate this object and prioritizing these potential owners. If the prediction is false for a user (she do not interact with the object), it gives the "ownership" of the object to the next user on the list of potential owners. Generally, this prediction is based on the position and the behavior of users (where they move, where they look, etc.).

In some applications, it may be useful to enable several users to manipulate a same object at the same time. Margery et al. [16] classify collaborative interactions into three categories:

- Only one user can manipulate an object at the same time. So the previous modes of concurrency control can be used.
- Several users can modify simultaneously independent parameters of a same object. So the previous modes of concurrency control can be adapted to each parameter of the objects.
- Several users can modify simultaneously co-dependent parameters of a same object. So another mechanisms has to be used to combine the users' actions and to modify appropriately the object.

4 Conclusion

We have presented a state of the art of architectures and mechanisms used in the CVE systems to maintain consistency of the virtual environment without disturbing the users' interactions. We have seen that the network architecture, but also the data distribution (i.e. where data is stored and executed on the network) have a strong impact on this consistency. Moreover, we have presented mechanisms which enables CVE systems to achieve a time synchronization between users or to manage the concurrent access to the data of the CVE. Table 1 presents a synthesis of the solutions used in some CVE systems to efficiently maintain consistency.

This state of the art report can help to determine the most adapted system architecture in order to address the requirements specific to a new CVE. However, this state of the art report shows also that the universal solution, which meets the requirements of all the CVE systems, does not exist yet.

We are currently involved in the French ANR project CollaViz and we are designing a new system architecture dedicated to collaborative scientific visualization.The requirements of this project impose several constraints about network architecture, security, and communications protocols. Indeed, we have to deal with a large amount of scientific data computed on a cluster, secured connections, standard networks, and mainstream hardware. So, this state of the art report may help us to find a good trade-off between the consistency of a CVE and the system responsiveness according to these requirements.

The large field of applications of this project encourages us to design flexible solutions. About data distribution, we expect to enable users to dynamically choose which model must be used. This choice will be made at the object level rather than at the application level because all the objects of the virtual environment have not the same need of consistency. About consistency maintenance, we expect to manage several groups of synchronization to enable all users to interact in the CVE event if they have hardware limitations such as low processing power and low network capabilities.

Acknowledgements

This work was supported in part by the French Research National Agency in the ANR-08-COSI-003-01 CollaViz project.

CVE	Network architecture	Data Distribution	Communication Protocols	Synchronization	Concurrency control
VISTEL [23]	client/server	centralized	unspecified	unspecified	none
RING [9]	client/server	homogenous replicated	UDP (*multicast*)	delayed global consistency	none (no concurrent access)
BrickNet [20]	client/server	partially replicated	UDP (*multicast*)	server synchronization	pessimistic
ShareX3D [11]	client/server	homogenous replicated	HTTP	server synchronization	pessimistic
SIMNET [4]	peer-to-peer	homogenous replicated	UDP (*broadcast*) + DIS	unspecified	none
DIVE [8]	peer-to-peer (+ a server saves the CVE state)	partially replicated	"MBone" (*multicast*)	time warp synchronization	none
OpenMask [15]	hybrid	homogenous replicated (uses the referent/proxies paradigm for the execution of object behaviors)	TCP (with PVM for OpenMASK3 and MPI for OpenMask4)	lockstep synchronization or delayed global consistency (possibility to specify the tolerated latency or to release the synchronization)	based on the referent/proxies paradigm (+ enables users to achieve simultaneous interactions)
SPLINE [21]	hybrid	partially replicated	*unicast* or *multicast* UDP, TCP, HTTP, RTP	unspecified	pessimistic
PaRADE [18]	hybrid	homogenous replicated	Unreliable *multicast* and reliable *multicast* using TCP	imposed global consistency	prediction based
Anthes et al. [1]	hybrid	homogenous replicated	*multicast* (based on a server hierarchy)	unspecified (imposed global consistency ?)	unspecified
ATLAS [12]	hybrid	partially replicated	*unicast* or *multicast*	unspecified	pessimistic, optimistic, and prediction based

Table 1: Synthesis of architectures and mechanisms used by some CVE systems for consistency maintenance

REFERENCES

[1] C. Anthes, P. Heinzlreiter, and J. Volkert. "An adaptive network architecture for close-coupled collaboration in distributed virtual environments". In *VRCAI'04: Proceedings of the 2004 ACM SIGGRAPH international conference on Virtual Reality continuum and its applications in industry*, pages 382–385, New York, NY, USA, 2004. ACM.

[2] C. Blanchard, S. Burgess, Y. Harvill, J. Lanier, A. Lasko, M. Oberman, and M. Teitel. "Reality built for two: a virtual reality tool". In *SI3D'90: Proceedings of the symposium on Interactive 3D graphics*, pages 35–36, New York, NY, USA, 1990. ACM.

[3] W. Broll. "DWTP - an Internet protocol for shared virtual environments". In *VRML'98: Proceedings of the third symposium on Virtual reality modeling language*, pages 49–56, New York, NY, USA, 1998. ACM.

[4] J. Calvin, A. Dickens, B. Gaines, P. Metzger, D. Miller, and D. Owen. "The SIMNET virtual world architecture". *IEEE Virtual Reality Annual International Symposium*, pages 450–455, Sep 1993.

[5] D. Delaney, T. Ward, and S. McLoone. "On consistency and network latency in distributed interactive applications: A survey – part I". *Presence: Teleoperators and Virtual Environments*, 15(2):218–234, 2006.

[6] D. Delaney, T. Ward, and S. McLoone. "On Consistency and Network Latency in Distributed Interactive Applications: A Survey – Part II". *Presence: Teleoperators and Virtual Environments*, 15(4):465–482, 2006.

[7] C. El Zammar. *"Interactions coopératives 3D distantes en environnements virtuels : gestion des problèmes réseau"*. PhD thesis, INSA de Rennes, 2005.

[8] E. Frcon and M. Stenius. "DIVE : A scaleable network architecture for distributed virtual environments". *Distrib. Syst. Engng.*, 5:91–100, 1998.

[9] T. A. Funkhouser. "RING: a client-server system for multi-user virtual environments". In *SI3D'95: Proc. of the symposium on Interactive 3D graphics*, pages 85–93, New York, NY, USA, 1995. ACM.

[10] C. Joslin, T. Di Giacomo, and N. Magnenat-Thalmann. "Collaborative virtual environments: from birth to standardization". *IEEE Communications Magazine*, 42(4):28–33, Apr 2004.

[11] S. Jourdain, J. Forest, C. Mouton, B. Nouailhas, G. Moniot, F. Kolb, S. Chabridon, M. Simatic, Z. Abid, and L. Mallet. "ShareX3D, a scientific collaborative 3D viewer over HTTP". In *Web3D'08: Proceedings of the 13th international symposium on 3D web technology*, pages 35–41, New York, NY, USA, 2008. ACM.

[12] D. Lee, M. Lim, S. Han, and K. Lee. "ATLAS: A Scalable Network Framework for Distributed Virtual Environments". *Presence: Teleoperators and Virtual Environments*, 16(2):125–156, 2007.

[13] M. Macedonia and M. Zyda. "A taxonomy for networked virtual environments". *IEEE Multimedia*, 4(1):48–56, Jan-Mar 1997.

[14] M. R. Macedonia, M. J. Zyda, D. R. Pratt, P. T. Barham, and S. Zeswitz. "NPSNET: A network software architecture for large scale virtual environments". *Presence*, 3(4):265–287, 1994.

[15] D. Margery, B. Arnaldi, A. Chauffaut, S. Donikian, and T. Duval. "OpenMASK: Multi-Threaded or Modular Animation and Simulation Kernel or Kit : a General Introduction". In *Virtual Reality International Conference (VRIC 2002)*, pages 101–110, 2002.

[16] D. Margery, B. Arnaldi, and N. Plouzeau. "A General Framework for Cooperative Manipulation in Virtual Environments". In *Virtual Environments'99*, pages 169–178, 1999.

[17] M. Mauve, V. Hilt, C. Kuhmunch, and W. Effelsberg. "A general framework and communication protocol for the transmission of interactive media with real-time characteristics". In *IEEE International Conference on Multimedia Computing and Systems (ICMCS'99)*, volume 2, pages 641–646, Jul 1999.

[18] D. Roberts and P. Sharkey. "Maximising concurrency and scalability in a consistent, causal, distributed virtual reality system, whilst minimising the effect of network delays". *Proceedings Sixth IEEE workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises*, pages 161–166, Jun 1997.

[19] C. Shaw and M. Green. "The MR Toolkit Peers Package and Experiment". In *IEEE Virtual Reality Annual International Symposium (VRAIS 93)*, pages 463–469. IEEE, 1993.

[20] G. Singh, L. Serra, W. Png, A. Wong, and H. Ng. "BrickNet: sharing object behaviors on the Net". *IEEE Virtual Reality Annual International Symposium (VRAIS 95)*, pages 19–25, Mar 1995.

[21] R. Waters, D. Anderson, J. Barrus, D. Brogan, S. Mckeown, T. Nitta, I. Sterns, and W. Yerazunis. "Diamond Park and Spline: A Social Virtual Reality System with 3D Animation, Spoken Interaction, and Runtime Modifiability". *Presence: Teleoperators and Virtual Environments*, 6(4):461–480, 1997.

[22] R. Waters, D. Anderson, and D. Schwenke. "Design of the Interactive Sharing Transfer Protocol". In *IEEE workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises*, pages 140–147, Jun 1997.

[23] M. Yoshida, Y. A. Tijerino, S. Abe, and F. Kishino. "A virtual space teleconferencing system that supports intuitive interaction for creative and cooperative work". In *SI3D'95: Proceedings of the symposium on Interactive 3D graphics*, pages 115–122, New York, NY, USA, 1995. ACM.